|  |
| --- |
|  |
|  |

**Cipher Craft:**

**Understanding and Implementing AES Encryption**

Group Members:

Yasar Azimi #44158

Shahzada Masood Asir #41311

Maryam Zakare #43203

Dr. Abdalrahman Salem Ali Alfagi

ITC 370: Intro to Cryptography and Data Security

***Final Group Project***

**December 14, 2023**

**Table of Contents**

**Introduction -------------------------------------------------------------------------------------------- Page 3**

**Main Body ---------------------------------------------------------------------------------------------- Page 3**

**Add Round Key --------------------------------------------------------------------------------------- Page 3**

**Substitute Key ----------------------------------------------------------------------------------------- Page 4**

**Shift Rows ---------------------------------------------------------------------------------------------- Page 6**

**Mix Columns ------------------------------------------------------------------------------------------- Page 7**

**Add Round Key ---------------------------------------------------------------------------------------- Page 8**

**Flow Chart ---------------------------------------------------------------------------------------------- Page 9**

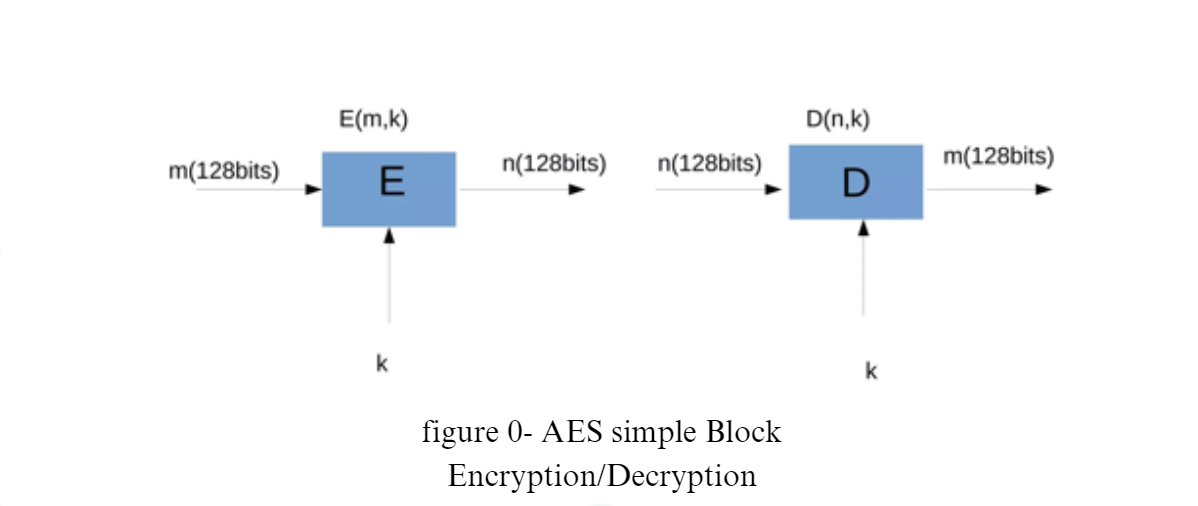
**Coding Section ------------------------------------------------------------------------------------------ Page 9**

**Conclusion and References--------------------------------------------------------------------------- Page 15**

1. **Introduction:**

AES (Advanced encryption standard), a symmetric block cipher adopted by NIST in 2001, was designed to replace DES as the approved standard for many applications. It is characterised by its complex structure and more substantial security base. AES is a private key symmetric block cipher, operating on 128-bit data with key lengths of 128, 192, or 256 bits, and it is more robust and faster than Triple-DES. The number of rounds in encryption and decryption depends on the key length:

1. 128-bit key: 10 rounds
2. 192-bit key: 12 rounds
3. 256-bit key: 14 rounds



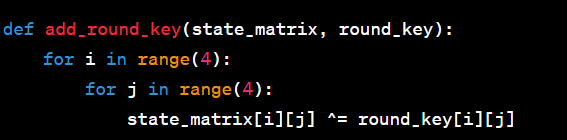
With real-world implementations ranging from VPNs to web browsers, AES is vital in securing digital communications. AES treats the 128 bits of a plaintext block as 16 bytes, allowing for efficient processing. AES is much more powerful and faster, which is why it’s used in WIFI, OS systems, Web browsers, and Mobile Applications. The number of encryption and decryption rounds varies based on the chosen key length, demonstrating the adaptability and robustness of the algorithm.

**Main Body:**

In this ambitious project, our comprehensive exploration of the AES (Advanced Encryption Standard) Transformation unfolds through five primary steps: Add Round Key, Substitute Bytes, Shift Rows, Mix Columns, and a final Add Round Key. Following the XOR operation in the initial step, the Substitute Bytes stage introduces non-linearity with a substitution table, enhancing cryptographic strength. Shifting Rows and Mixing Columns contribute to diffusion and confusion, ensuring each byte's influence spans multiple bytes. The provided flowchart visually depicts the logical flow, and code snippets offer a practical understanding, underscoring the intricacies of this vital encryption process.

1. **Add round key**:

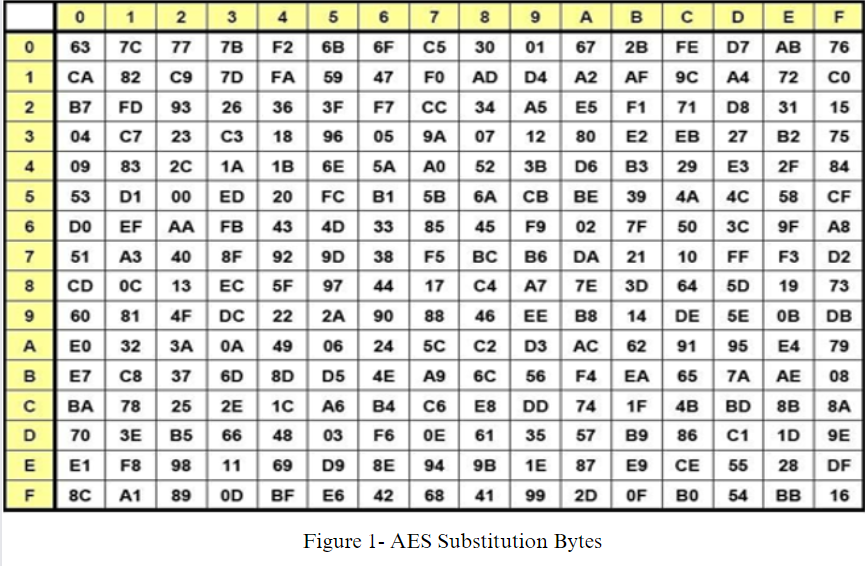
In this crucial phase, each byte of the state matrix undergoes XOR with the corresponding byte from the round key. This pivotal process bolsters security by integrating the round key, derived from the original key, into the state matrix. Essentially, it entails XORing each byte of the state matrix with the corresponding byte of the round key, where the round key is derived from the main or original encryption key through a key schedule, matrix, or algorithm.



1. **Substitute bytes**:

In this pivotal phase, every byte within the state matrix undergoes substitution with its corresponding value from the S-Box, a predefined substitution table. This substitution introduces non-linearity to the algorithm, thereby fortifying its resilience against cryptographic attacks. A 16 x 16 matrix of byte values, known as an S-box, is employed in this process. This S-box encompasses a permutation of all possible 256 8-bit values, contributing significantly to the robustness of the encryption process. Each individual byte of State is mapped into a new byte where:

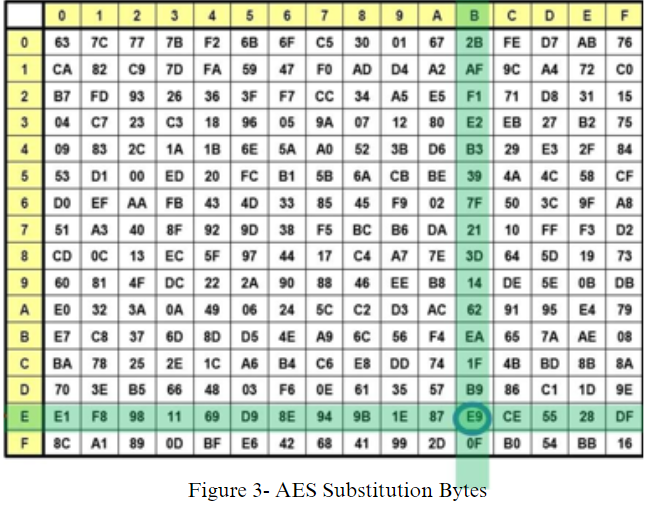
1. Leftmost 4 bits of the byte 🡪 Row value
2. Rightmost 4 bits of the byte 🡪 column value



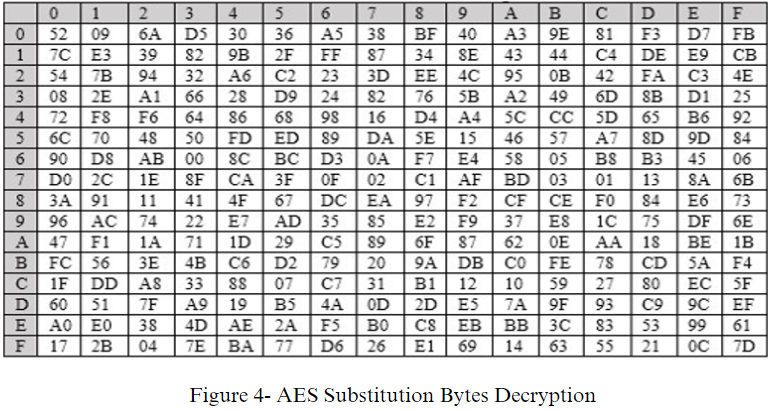
**Example:**

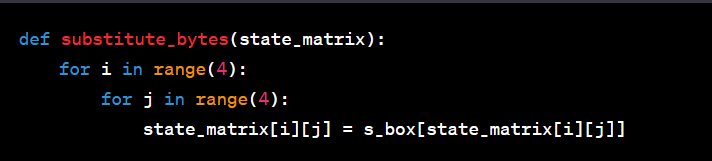
🡪  🡪

Figure 2 – Substitution byte examples 1



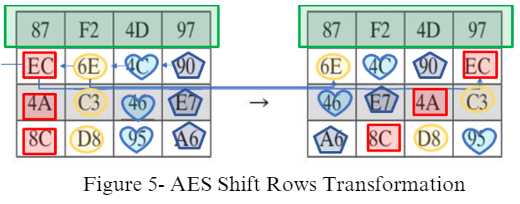
For the Decryption process we use the inverse S-box while the operation is the same.



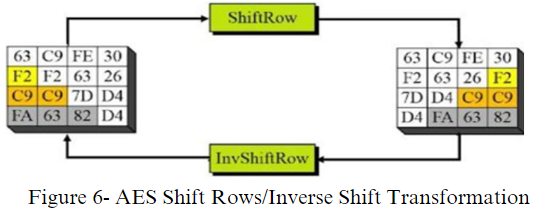


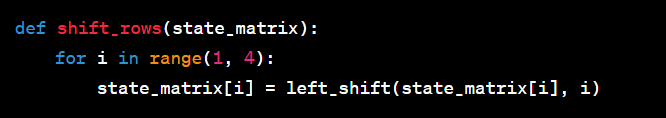
1. **Shift rows**

In this particular part, every row in the state matrix undergoes a leftward shift by a specified offset, where the row index determines the offset. The first row remains unshifted, the second row shifts by one position to the left, the third row by two positions, and the fourth row by three positions. This row-shifting mechanism ensures that each byte influences multiple bytes in the subsequent round, thereby enhancing diffusion and contributing to the overall security of the algorithm.



In the decryption process, we employ the inverse shift row transformation, known as InvShiftRows. In these inverse transformations, the first row remains unshifted, the second-row shifts by one position to the right, the third-row shifts by two positions to the right, and the fourth-row shifts by three positions to the right. This inverse shifting mechanism ensures the reversal of the original shift row transformation, contributing to the accurate retrieval of the plaintext during decryption.





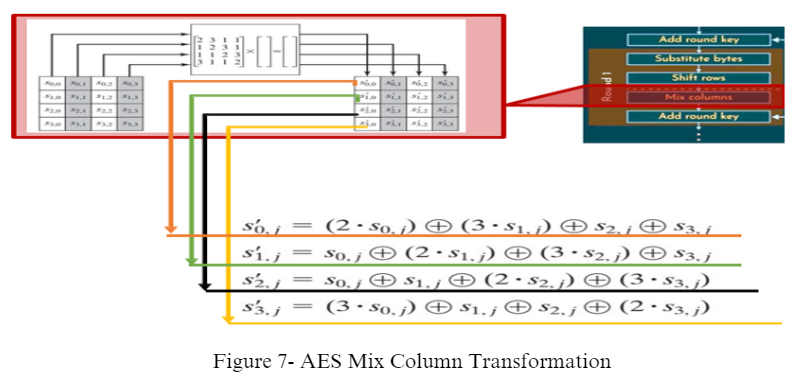
1. **Mix columns:**

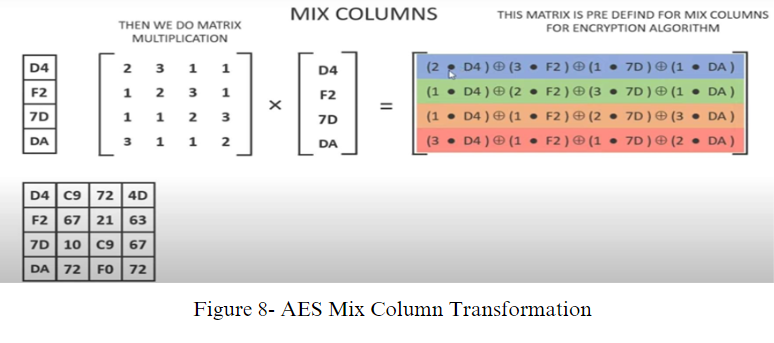
This stage introduces mixing and diffusion by combining the values in each column through matrix multiplication. It performs a Mix Columns operation on the state matrix using fixed matrix multiplication. Each column is treated as a polynomial and multiplied in the Galois Field. Like row shifting, this operation has both a forward and an inverse transformation.

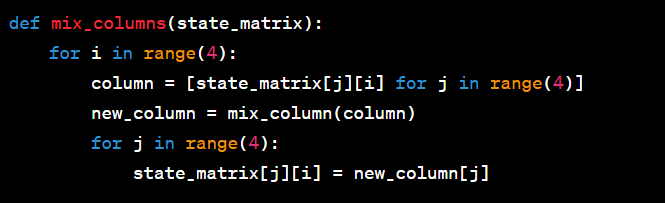
A) The forward MixColumns transformation, known as MixColumns Forward Transformation:

a) Each byte of a column is mapped to a new value that is a function of all four bytes in that column.

b) The transformation can be defined by the matrix multiplication on the state matrix.



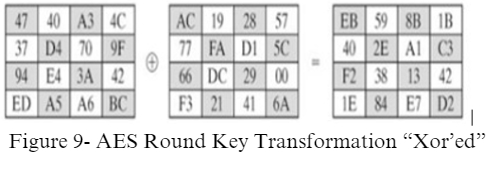


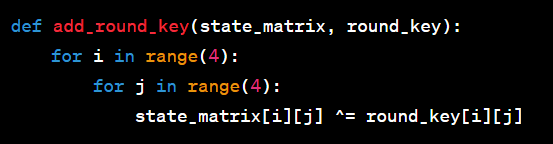


1. **Add round key**

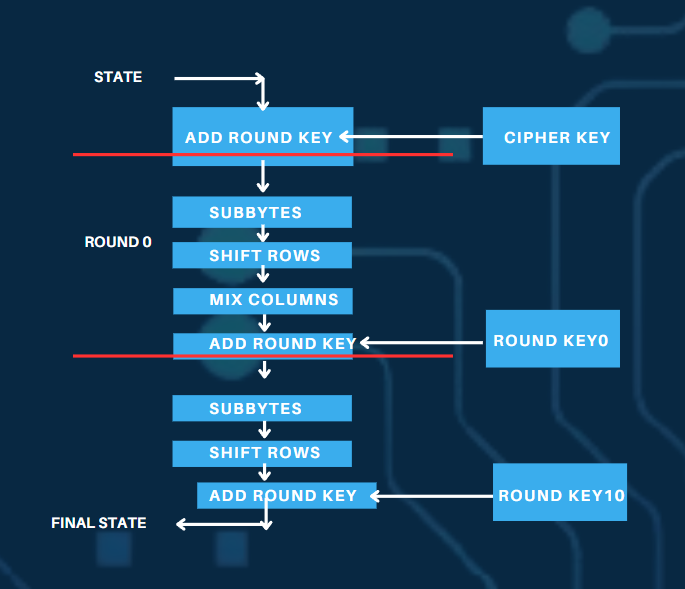
The ultimate application of the round key guarantees that each byte in the state matrix is influenced by the key at least twice, thus enhancing security. Similar to Step 1, XOR each byte of the state matrix with the corresponding byte of the round key. Additionally, there are both Forward and Inverse Transformations for this round, ensuring a comprehensive and reversible cryptographic process.

1. In the forward add round key transformation, called AddRoundKey, the 128 bits of State are bitwise XORed with the 128 bits of the round key.





**Flow chart:** This flowchart represents the logical flow of the AES single-round implementation.



**AES Program Coding**

This Python code presents a user-friendly graphical interface designed for the encryption and decryption of plain text through the application of the Advanced Encryption Standard (AES) algorithm, specifically in Electronic Codebook (ECB) mode. The script utilizes the Tkinter library to construct the graphical user interface, ensuring an intuitive user experience. Additionally, for the implementation of cryptographic functions, it leverages the cipher module from the Cryptodome library, which enhances the security aspects of the encryption and decryption processes.

The Tkinter library is employed to create an interactive and visually appealing interface, allowing users to easily interact with the encryption and decryption functionalities. Through the integration of Cryptodome's cipher module, the script ensures robust cryptographic operations, reinforcing the security of the plaintext transformation. The combination of Tkinter and Cryptodome libraries not only facilitates a seamless user experience but also underscores the commitment to employing reliable cryptographic techniques for data security within the Python script.

**Dependencies:**

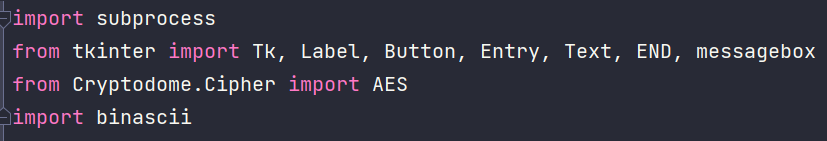
Python 3.x

Tkinter (usually bundled with Python)

PyCryptodome

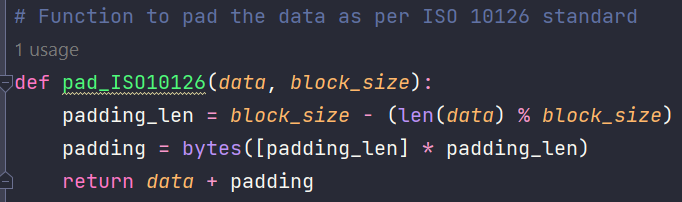
Install PyCryptodome using pip:

pip install pycryptodome

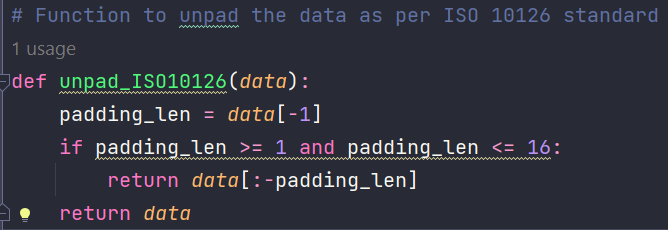


**Functions:**  **Padding and Un-padding Functions with ISO 10126 Standard:**

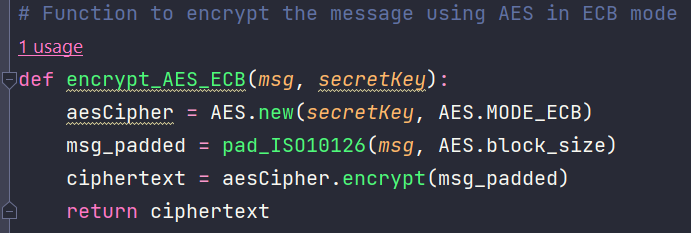
To ensure compliance with the ISO 10126 standard, the pad\_ISO10126 function is implemented to pad data according to specified requirements. This function takes two parameters: data, which represents the data to be padded, and block\_size, indicating the block size of the cipher.



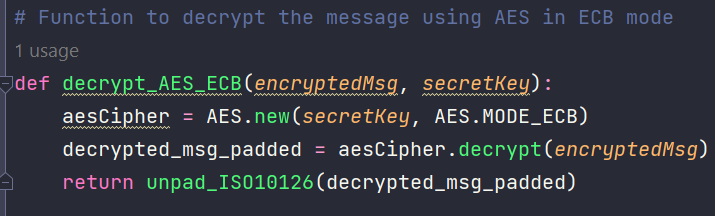
Similarly, the unpad\_ISO10126 function is designed to remove padding from previously padded data. It takes the data parameter, representing the padded data. These functions collectively provide a reliable means of padding data according to ISO 10126 standards and subsequently removing the padding as needed. The implementation details within the functions ensure proper adherence to the specified padding standard.



**Encryption and Decryption: AES Encryption and Decryption in ECB Mode**

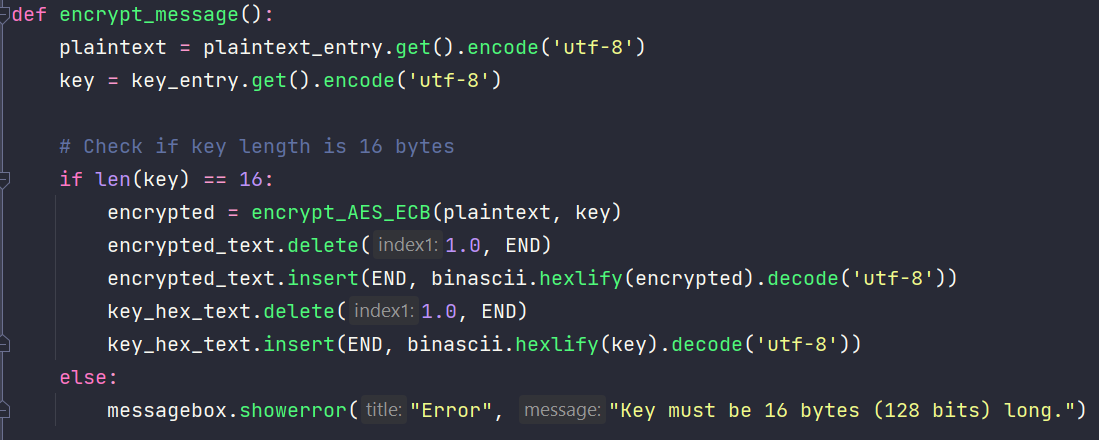
For encrypting a message using AES in ECB mode, the encrypt\_AES\_ECB function is defined. This function takes two parameters: msg, representing the plaintext message, and Secret-Key, which is the encryption key (16 bytes) used for the encryption process. 

Conversely, the decrypt\_AES\_ECB function is crafted to decrypt a message that has been encrypted using AES in ECB mode. It takes two parameters: encryptedMsg, the encrypted message, and Secret-Key, the encryption key used during encryption. These functions together provide a convenient and secure way to encrypt and decrypt messages using the AES algorithm in ECB mode. The implementation details within the functions ensure proper handling of encryption and decryption processes while maintaining the specified key lengths and mode of operation.

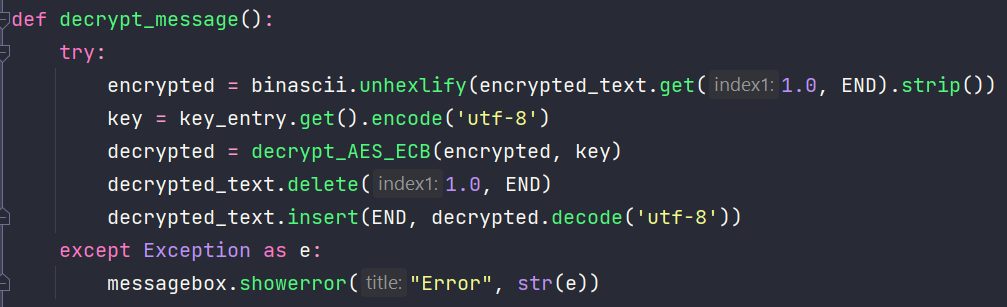


**Encryption, Decryption:**

The encrypt\_message function is designed to manage the encryption process within the graphical user interface (GUI). It handles the encryption of a message using the specified encryption algorithm and parameters.

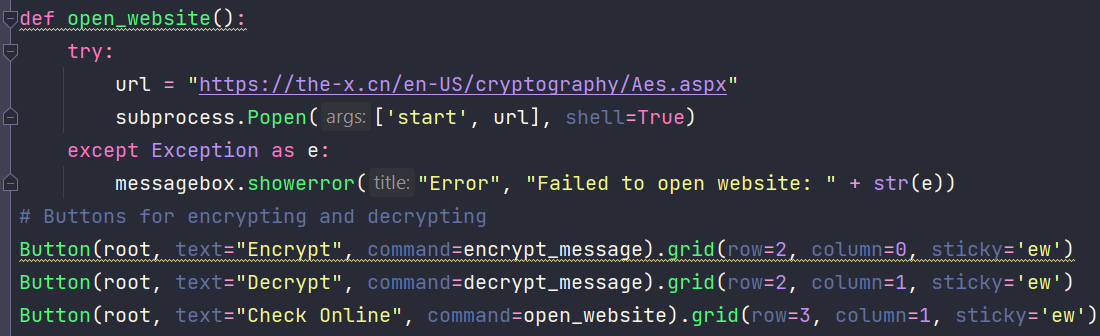


Similarly, the decrypt\_message function oversees the decryption process within the GUI. It facilitates the decryption of an encrypted message, ensuring that the appropriate decryption algorithm and parameters are applied.



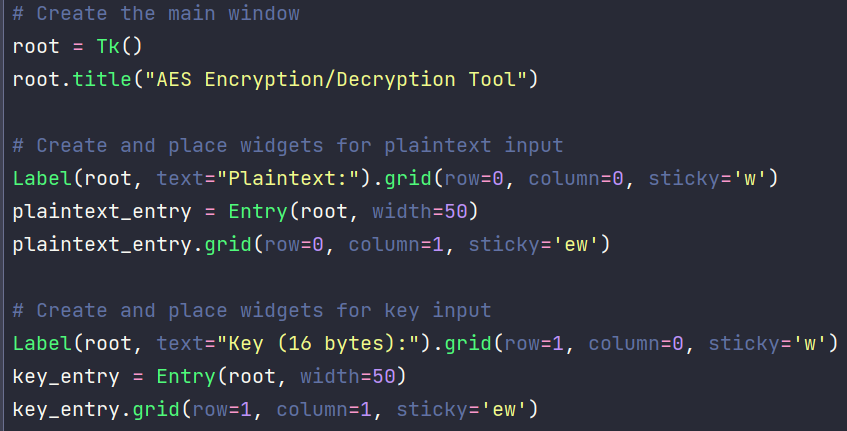
**GUI-Related Functions:** URL Opening Functions:

Additionally, the open\_website function is designed to open a specified URL in a web browser. This function takes care of launching the default web browser and navigating to the provided URL. These functions collectively provide a clear and modular approach to managing encryption, decryption, and URL opening processes within a GUI application. The implementation details within each function can be customized based on the specific requirements of the application.



**GUI Components Main Window:**

Created using Tk(). Labels: To indicate where to input plaintext, key, and to show encrypted/decrypted messages. Entry Widgets: For input of plaintext and the encryption key. Buttons: For triggering encryption, decryption, and opening a website. Text Areas: For displaying the encrypted message, decrypted message, and the key in hexadecimal format.



**Layout**

The layout is arranged in a grid with specific rows and columns for each component. Labels, entry widgets, buttons, and text areas are strategically placed for user convenience.



**Error Handling**

Checks for the correct length of the encryption key (16 bytes). Handles decryption errors and displays messages using a message box.

**Customization**

The encrypted text area is pre-filled with a sample message. A credit label for the creators is included at the bottom of the window.

**Running the Tool**

Ensure all dependencies are installed. Run the script using a Python interpreter. Use the GUI to input text, keys, and perform encryption or decryption operations.

**b. Conclusion:**

Implementing a single round of the Advanced Encryption Standard (AES) involves key steps: Add Round Key, Substitute Bytes, Shift Rows, Mix Columns, and a final Add Round Key. These steps collectively introduce non-linearity, confusion, and diffusion into the encryption process. The Add Round Key step XORs the state matrix with the encryption key, creating a unique combination. Substitute Bytes replace each byte with values from a substitution table, enhancing non-linearity. Shifting Rows and Mixing Columns introduce diffusion through horizontal and mathematical operations.

The final Add Round Key operation ensures that key material is recombined, reinforcing encryption. Note that the code snippets are illustrative, and a complete implementation requires additional functions and constants. AES's strength lies in combining these steps across multiple rounds, providing a secure encryption scheme.

**NOTE: The link for the video:**

**https://drive.google.com/file/d/1s6G5G\_103r6U0FyCML\_vPBW0Fw\_ukwMo/view?usp=sharing**

**NOTE: The Link to the Google Drive is here: The drive has the Documentation, Presentation, Video and the python File of software.**

**https://drive.google.com/drive/folders/140LQvOPtvfKWl6o1ogz7xa\_23J9xsRAT?usp=drive\_link**

**References:**

* Stallings, W. (2016). Cryptography and Network Security: Principles and Practice, 7th Global Edition. Pearson Higher Education.